Транзакции

При проектировании информационных систем возникают ситуации, когда сбой в системе или какой-либо ее периферийной части может привести к утрате информации или к финансовым потерям. Простейшим примером может служить ситуация с перечислением денег с одного счета на другой. Если сбой произошел в тот момент, когда операция снятия денег с одного счета уже произведена, а операция зачисления на другой счет еще не произведена, то система, допускающая такие ситуации, должна быть признана не отвечающей требованиям заказчика. Или должны выполняться обе операции, или не выполняться вовсе. Такие две операции трактуют как одну и называют транзакцией.

Транзакция, или деловая операция, определяется как единица работы, обладающая свойствами ACID:

- *Атомарность* — две или более операций, выполняются все или не выполняется ни одна.
- *Согласованность* — при возникновении сбоя система возвращается в состояние до начала неудавшейся транзакции. Если транзакция завершается успешно, то проверка согласованности удостоверяется в успешном завершении всех операций транзакции.
- *Изолированность* — во время выполнения транзакции все объекты-сущности, участвующие в ней, должны быть синхронизированы.
- *Долговечность* — все изменения, произведенные с данными во время транзакции, обязательно сохраняются, например, в базе данных, что позволяет восстанавливать систему.

Для фиксации результатов работы SQL-операторов, логически выполняемых в рамках некоторой транзакции, используется SQL-оператор **COMMIT**. В API JDBC эта операция выполняется по умолчанию после каждого вызова методов **executeQuery()** и **executeUpdate()**. Если же необходимо сгруппировать запросы и только после этого выполнить операцию **COMMIT**, сначала вызывается метод **setAutoCommit(boolean param)** интерфейса **Connection** с параметром **false**, в результате выполнения которого текущее соединение с БД переходит в режим неавтоматического подтверждения операций. После этого выполнение любого запроса на изменение информации в таблицах базы данных не приведет к необратимым последствиям, пока операция **COMMIT** не будет выполнена непосредственно. Подтверждает выполнение SQL-запросов метод **commit()** интерфейса **Connection**, в результате действия которого все изменения таблицы производятся как одно логическое действие. Если же транзакция не выполнена, то методом **rollback()** отменяются действия всех запросов SQL, начиная от последнего вызова **commit()**. В следующем примере информация добавляется в таблицу в режиме действия транзакции, подтвердить или отменить действия которой можно, снимая или добавляя комментарий в строках вызова методов **commit()** и **rollback()**.

Для транзакций существует несколько типов чтения:

- грязное чтение (*dirty reads*) происходит, когда транзакциям разрешено видеть несохраненные изменения данных. Иными словами, изменения, сделанные в одной транзакции, видны вне ее до того, как она была сохранена. Если изменения не будут сохранены, то, вероятно, другие транзакции выполняли работу на основе некорректных данных;
- неповторяющееся чтение (*nonrepeatable reads*) происходит, когда транзакция А читает строку, транзакция Б изменяет эту строку, транзакция А читает ту же строку и получает обновленные данные;
- фантомное чтение (*phantom reads*) происходит, когда транзакция А считывает все строки, удовлетворяющие **WHERE**-условию, транзакция Б вставляет новую или удаляет одну из строк, которая удовлетворяет этому условию, транзакция А еще раз считывает все строки, удовлетворяющие **WHERE**-условию, уже вместе с новой строкой или недосчитавшись старой.

JDBC удовлетворяет уровням изоляции транзакций, определенным в стандарте SQL:2003.

Уровни изоляции транзакций определены в виде констант интерфейса **Connection** (по возрастанию уровня ограничения):

- **TRANSACTION_NONE** — информирует о том, что драйвер не поддерживает транзакции;
- **TRANSACTION_READ_UNCOMMITTED** — позволяет транзакциям видеть несохраненные изменения данных, что разрешает грязное, неповторяющееся и фантомное чтения;
- **TRANSACTION_READ_COMMITTED** — означает, что любое изменение, сделанное в транзакции, не видно вне ее, пока она не сохранена, что предотвращает грязное чтение, но разрешает неповторяющееся и фантомное;
- **TRANSACTION_REPEATABLE_READ** — запрещает грязное и неповторяющееся чтение, но фантомное разрешено;
- **TRANSACTION_SERIALIZABLE** — определяет, что грязное, неповторяющееся и фантомное чтения запрещены.

Метод **boolean supportsTransactionIsolationLevel(int level)** интерфейса **DatabaseMetaData** определяет, поддерживается ли заданный уровень изоляции транзакций.

В свою очередь, методы интерфейса **Connection** определяют доступ к уровню изоляции:

int getTransactionIsolation() — возвращает текущий уровень изоляции;

void setTransactionIsolation(int level) — устанавливает необходимый уровень.